DOCUMENT RESUME

ED 160 126 IR 006 611

| | |
|---|---|
| TITLE | A Functional Specification for a Programming Language for Computer Aided Learning Applications. |
| INSTITUTION | National Research Council of Canada, Ottawa (Ontario). |
| REPORT NO | NRCC 13659 |
| PUB DATE | Jul 72 |
| NOTE | 52p. |
| AVAILABLE FROM | Secretary, Associate Committee on Instructional Technology, National Research Council of Canada, Ottawa, Ontario, Canada, K1A OR6 (Free, while supply lasts) |
| EDRS PRICE | MF-$0.83 HC-$3.50 Plus Postage. |
| DESCRIPTORS | Bibliographies; *Computer Assisted Instruction; Criteria; Needs Assessment; *Performance Specifications; Programing; *Programing Languages |

ABSTRACT

In 1972 there were at least six different course authoring languages in use in Canada with little exchange of course materials between Computer Assisted Learning (CAL) centers. In order to improve facilities for producing "transportable" computer based course materials, a working panel undertook the definition of functional requirements of a user oriented programming language for computer aided learning applications. The introduction of this report, which outlines the scope, is followed by areas of application of the language and its general properties. Levels of implementation include: display information, accept user input, process user input, identify locations in program, sequence execution, record and manipulate data, user tools available within the language, coding conventions, and a guide to preparation of detailed specification and implementation. This report was prepared by a Working Committee on CAL Languages for the National Research Council's Associate Committee on Instructional Technology; the appendix lists documents presented to the working panel. (VT)

**********
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
**********

ED160126

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

# A FUNCTIONAL SPECIFICATION FOR A PROGRAMMING LANGUAGE FOR COMPUTER AIDED LEARNING APPLICATIONS

National Research Council of Canada
Associate Committee on Instructional Technology
Ottawa 1972

NRCC 13659

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Jean-Yves Fortin

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC) AND USERS OF THE ERIC SYSTEM."

IR006611

# A Functional Specification for a Programming Language for Computer Aided Learning Applications

## *Foreword*

This Functional Specification was prepared for the National Research Council's Associate Committee on Instructional Technology by a Working Panel on CAL Languages whose membership was drawn from groups across Canada actively engaged in Computer Aided Learning Research and Development.

The Associate Committee has made the Specification available for publication in order to solicit critical comment prior to its anticipated implementation and its recommendation for adoption as a national standard. Comments and suggestions should be forwarded to:

The Secretary
Associate Committee on Instructional Technology
National Research Council of Canada
Montreal Road
Ottawa, Canada K1A OR8

CONTENTS PAGE

List of Members of Working Panel on CAL Languages . . . 1

Acknowledgement . . . . . 3

Introduction . . . . . 4

Areas of Application of the Language . . . . . 5

General Properties of the Language . . . . . 5

Levels of Implementation . . . . . 7

1. Display Information . . . . . 10

2. Accept User Input . . . . . 15

3. Process User Input . . . . . 19

4. Identify Locations in Programme . . . . . 28

5. Sequence Execution . . . . . 30

6. Record and Manipulate Data . . . . . 35

7. User Tools Available within the Language . . . . 39

8. Coding Convenience . . . . . 40

9. Guide to Preparation of Detailed Specification and Implementation . . . . . 42

Appendix A Documents Presented to the Working Panel. . 46

*Working Panel on CAL Languages*

Mr. J. W. Brahan (Chairman)
Secretary, Associate Committee on Instructional Technology
National Research Council of Canada
Montreal Road
OTTAWA, Ontario K1A OR8

Dr. Yves Bégin
Institut de recherche pédagogique
914, Monseigneur-Grandin
STE-FOY, Québec

Dr. K. G. Brown
Director of Graduate Studies
Faculty of Education
The University of New Brunswick
FREDERICTON, New Brunswick

Dr. Stacy Churchill *
Assistant Coordinator
Office of Research and Development Studies
The Ontario Institute for Studies in Education
252 Bloor Street West
TORONTO 5, Ontario

Dr. Herbert J. Hallworth
Professor of Educational Psychology
Faculty of Education
The University of Calgary
CALGARY 44, Alberta

Dr. Steve Hunka
Coordinator
Division of Educational Research
Faculty of Education
The University of Alberta
EDMONTON 7, Alberta

M. François Labrousse
Service de l'Informatique
Ministère de l'Education
265, avenue de la Couronne
QUEBEC, Québec

Dr. Stephen K. Lower **
Assistant Professor of Chemistry
Simon Fraser University
BURNABY 2, British Columbia

Dr. Eric D. MacPherson
Chairman
Mathematics Education Department
The University of British Columbia
VANCOUVER 8, British Columbia

Mrs. Rosemary Newkirk
CAI Coordinator
Department of Computer Science
The University of Western Ontario
LONDON 72, Ontario

Dr. Gale H. Roid
Assistant Professor
Centre for Learning and Development
McGill University
P.O. Box 6070
MONTREAL 101, Quebec

- - - - - - - - - - - - - - - - -

* Alternate: Dr. R. S. McLean
** Alternate: Mr. Brian Funt

*Acknowledgment*

The Working Panel relied heavily on the 1969 EDUCOM report* of Professor Karl L. Zinn. The aspects which he used to compare programming languages have been used by the Panel as the basis for the definition of language requirements contained in this functional specification.

The Panel is grateful to the many organizations who forwarded information regarding their use of programming languages for instructional applications.

* Karl L. Zinn, "Comparative Study of Languages for Programming Interactive Use of Computers in Instruction". EDUCOM Research Memorandum RM-1469, February, 1969.

*Introduction*

A major obstacle to the exchange of computer-based curriculum materials among various installation results from the multiplicity of course authoring languages which exist. In Canada, there are currently at least six different course authoring languages in use with little exchange of course materials between computer aided learning centres. With the view of improving facilities for producing "transportable" computer based course materials, the Associate Committee on Instructional Technology established a Working Panel on CAL Languages. The task of this Working Panel was to define the functional requirements of a user oriented programming language for computer aided learning applications, with particular emphasis being placed on satisfying requirements peculiar to Canada. Membership of the Working Panel was drawn from groups across Canada actively engaged in CAL activities.

The requirements as defined by this functional specification represent a definition of the capabilities of the language from the user point of view. Details of how these capabilities are achieved would be defined in a detailed specification based on the requirements as stated in this document. In this specification, where examples are used to illustrate particular functions, it should be noted that these examples are for the purpose of illustration only, and should in no way be interpreted as a definition of the format of the language.

In specifying the requirements of the language, consideration has been given to the current state-of-the-art and related economics. However, if it is indicated that the realization of any of the required functions would result in unreasonable cost, or complexity whether in the implementation of the language or in its use, the Secretary, Associate Committee on Instructional Technology should be informed accordingly at the earliest possible stage.

*Areas of Application of the Language*

Within the general area of "Computer-Assisted Instruction" (CAI), the language should provide a facility for such applications as:

- Interactive Tutorial instruction (including drill and practice)
- Operation as a problem solving tool
- Simulation
- Inquiry or Information Retrieval

In all of the above applications, consideration must be given to use of the facility by student, teacher, author, and researcher.

The term Computer Aided Learning (CAL) should be interpreted to include a wide range of computer applications in instructional systems. In general, these applications require the language to provide a flexible facility for recording and retrieving data on students relevant to instructional decision making, for sorting these data and for performing elementary statistical processing of such data. These data would include both individual data as well as group data. Of prime importance is the facility provided by the language for the presentation of instructional materials and the acceptance and analysis of responses. Furthermore, provision should be made for the facility to interface in a complementary fashion with other types of instructional techniques, both regular classroom instruction and related computer-oriented types of instruction such as CMI and CBI.

*General Properties of the Language*

These general properties of the language are to be used as a guide in interpreting the more detailed

statement of functional requirements which follows.

- The language should provide capabilities to permit effective communication with the computer through a wide range of input/output devices by authors, students, teachers, and researchers. Such communication may be by means of an on-line interactive terminal, or it may be by means of off-line batch processing which would have applications in "computer-managed instruction" operations, and also as related to analysis of results obtained during interactive instructional use. Access to extensive input-output facilities is of utmost importance in computer-aided-learning applications. The language must provide the user with a means whereby he can make effective use of both conventional computer input-output devices as well as devices designed specifically for CAL applications. In many cases, the standard keyboard driven alphanumeric display is a completely ineffective device for communicating with the user. One need only think of applications involving young children who cannot read. In some cases, the user may have a perceptual handicap, requiring special consideration insofar as the computer input-output facility is concerned. Thus the language must be capable of coping with a wide range of input-output devices and provide the user with effective access to these devices in a variety of applications.

- The language must be easy to learn and to use by non-computer oriented users. It should provide a means whereby the relatively unsophisticated computer-user can gain effective access to the full computing power available for instructional applications of the computer. One approach which should be considered for providing quick and meaningful access by the novice would be to structure the language in such a way that it is not necessary to learn the whole language before being able to use it effectively.

- The language should incorporate extensive facilities for the author to assist in coding course materials; testing, debugging and updating of these materials; defining information to be recorded; and in analysing the recorded results which are obtained.

The command structure of the language must be such that it is applicable in both English and French. All answer processing facilities incorporated in the CAL language should take into account the requirements of both languages. Such answer processing facilities must be incorporated in such a way as to be readily expandable.

The format and command structure of the language should be organized so as to promote "built-in" documentation and hence make programmes written in the language easy to follow.

It should be noted that, in many instances, the language may be implemented as part of a general time-sharing system and consideration should be given to current practices which apply in the operation of such systems.

The language must be structured so as to facilitate the incorporation of new features as they are required. These new features include software for extending the processing and record gathering capabilities of the language as well as software required to make effective use of new input-output devices as they become available in the future. The design of the language must not in any way limit its applicability to currently available hardware.

*Levels of Implementation*

In addition to the full implementation of the language, it is recognized that there may be applications

which do not require all of the features of the language. Thus a basic subset of the language has been specified. This basic subset has been further divided to take into account those applications which require no extended input-output facilities. An experimental level has also been recommended which would be used for assessment of facilities which might be added to the language. In all cases, upward compatibility is a necessity.

It is recognized that implementation of the language may be in the form of "primitive functions" and "derived functions". In such cases, there might be little difference between the capabilities of different levels; the higher level simply has more functions provided as primitives than the lower level or subset. However, it was felt that it would be useful to indicate a minimum subset of the language in that it serves to define the minimum demands which would be made on the system on which it is implemented.

Where a facility is specified as being required at a given level, it is not sufficient that the language simply be capable of carrying out the function specified. If the facility requires extensive complex programming to invoke, it does not exist for the majority of potential users. In order for a facility to be usable, it is necessary to be able to use it with little or no effort. Some facilities which are used by a great majority of users may, in fact, be made automatic with the option being available to the user of "turning the facility off". (Examples of facilities of this nature might include: elimination of multiple spaces and automatic conversion to upper case, as required, when the terminal has no facilities for lower case).

The proposed levels of implementation are:

Level 2: This level should be considered the "full set" of the language. Implementation of the language at this level would include all of the proven features of the language. Any programmes prepared using the subset of the language (either level 1 or level 0) would run on implementations at this level.

Level 1: This level should be considered the basic subset of the language. It contains the minimum required processing capabilities and permits extensive use of input-output facilities.

Level 0: This is the lowest level of implementation and should only be used in applications where input-output is limited to keyboard driven character display devices. All of the processing capabilities of level 1 will be included at this level, only the input-output handling capabilities will be reduced.

Level X: This level should be considered an experimental level of implementation. Its prime purpose would be to test features which might be approved for implementation in the operational level(s) of the language. It would include all of the features of level 2 of the language as well as "experimental" features. It would normally be installed on systems being used to support the language. Any programmes written in this level of the language must carry a warning regarding the experimental features required.

## 1. *Display Information*

### 1.00 General

The language should permit the display of:

- Pre-stored information, held in an external device, such as a slide projector, tape recorder, or other information storage device capable of being operated under control of the computer.
- Pre-stored information, held in the internal store of the computer system.
- Generated information, produced according to the rules of user defined algorithms.
- Generated information produced by combining units of information in various ways.

The available display equipment is the only means whereby the system can present information to the student, teacher, researcher, or author and react to their input. Thus it is of utmost importance that the language include facilities for coping with the range of display equipment required for a wide variety of subject matter as well as a variety of applications within the field of Computer Aided Learning. Details of the language specification shall be structured so that new display devices which may be available in the future can readily be assimilated into the language in such a way that effective use can be made of them.

It is a requirement that it be possible to reference all output devices by logical name.

*Level*

1.01 Display Characters

Display characters on a typewriter, CRT, or other computer-controlled display device. Necessary facilities shall be included to permit flexible formatting of alphanumeric information on such display devices. *0*

1.01.1 Basic Character Set

Text information should be defined using a basic coded character set which must include facilities for displaying accented characters required for French language text. The basic coded character set shall include both upper and lower case characters and shall conform to applicable Canadian and international standards. It is to be noted that many commonly used computer terminals do not have lower case capabilities and the use of such terminals must be possible without requiring complex procedures. *0*

1.01.2 Special Character Sets

Special character sets should be defined as extensions of the basic code set. Such special character sets should be accessed using the procedures as defined in the appropriate ISO and/or CSA standards documents. Such special character sets should include facilities for subscripts and superscripts. Any special character sets which are not defined to the language in this way must have their complete definition included as part of the programme in which they are used. *1*

*Level*

1.02 Display Graphics

1.02.1 Display Predefined Graphics

Display graphics (including characters as a special case) on CRT, slide or film projector and screen, video tape player, or other computer controlled device. These patterns may have been previously defined and stored in central or peripheral files. *1*

1.02.2 Display User-Defined Graphics

Display graphics generated as required using a stored procedure and current parameter values provided by the user. *1*

1.02.3 Display Large Characters

As a special case of 1.02 it shall be possible to display the character sets defined in 1.01, in large font. Several sizes (at least 3) should be selectable and these should be readily accessible to the author without complex procedures being required to invoke the facility. *2*

1.03 Display Audio

1.03.1 Pre-Defined Audio

Play pre-recorded addressable audio messages on speaker or headphones. Such audio messages may have been pre-recorded by the user "off-line". *1*

1.03.2 User-Defined Audio

Play audio messages assembled, or generated from specifications provided by the user. *X*

*Level*

1.04 Display Control Functions

1.04.1 Time Display

Time display, i.e., pause before continuing with next display or before accepting user input. Although the resolution requirements may be re-specified in light of future needs, a resolution of 1/4 second is required at level 2 with coarser resolution being acceptable at levels 0 and 1. (It is recognized that resolution may be limited by the mode of operation of any particular computer system on which the language is implemented). *0*

1.04.2 Define Spatial Position of Material Displayed

Define spatial position through coordinates, relative distance from reference points or previously defined material, or format control characters of the input device.

1. - The minimum definition would be based on the number of lines and the number of character spaces from a given reference position. (At level 0, reverse line feed would be prohibited). *0*

2. - A more advanced definition would be based on an X-Y grid system of moderate resolution. *1*

3. - It is required that it be possible for the user to specify the spatial position by calculation at execution time. *1*

4. - It shall be possible for the user to label areas of the display by assignment and/or by calculated values. *1*

1.04.3 Modify Existing Display

Erase selectively, write over (overlay) or add. *1*

*Level*

1.04.4 Assemble Material at Time of Display

Compose, from files of elements according to rules or statements of relationship provided with the instruction to display material. Such files of elements may consist of:

1. Text *0*
2. Graphics *1*

It is to be expected that more extensive assembly facilities would be provided at level 2 than at levels 0 and 1.

1.05 Experimental Display Control Functions

The following display control functions are considered to be experimental and are recommended for inclusion in implementations of the language at level X.

1.05.1 Coordinate Modes of Display

Coordinate different display modes or devices such as film, audio, and typewriter. A sophisticated example would be the ability to synchronize separate audio and video sources. Limited facilities for such functions should be included at level 2. *X*

1.05.2 Combine Displays

Superimpose one visual display upon another, e.g., graph on projected image. *X*

1.05.3 Define Motion

Define a dynamic display which when used will vary over time, as a bouncing ball or representation of fluid flow. *X*

*Level*

1.05.4 Control Special Devices

Supply appropriate control signals to special purpose devices, e.g., analogue output to an X-Y recorder. X

2. *Accept User Input*

2.00 General

- The language shall include facilities for accepting input from a wide range of devices. These would include, in addition to the traditional computer input devices, a variety of special purpose input equipment used to provide effective access to the computer for the non-computer oriented user (student, teacher, or author). Consideration must be given in the structure of the language to future incorporation of input devices which will be developed.

- The language shall permit the user to take full advantage of the facilities offered by specific input devices without having to occupy himself with detailed handling routines for such devices.

- The implementation of input facilities applicable to keyboards shall, wherever possible, be in such a way as to keep to a minimum the requirement for multiple key operations. (Specific functions to which this particularly applies include: character deletions, enter operations, backspace, and character set change.)

- As with all external devices, it is a requirement that it be possible to refer to input devices using a logical name.

*Level*

2.01 Accept Input Information

2.01.1 Character Input

Accept characters from keyboard or special purpose keyset. Input formats should be as flexible as possible. In particular, it is desirable that it be possible to treat multiple lines of input as a single response. *0*

2.01.2 Graphic Input

Accept graphics (e.g., coordinates of a point, line, or other pattern) from light pen, tablet stylus, or other user operated graphic encoding device. It should be possible to automatically relate a graphic input point to display areas as defined in 1.04.2.4. (See 3.02.4.1). *1*

2.01.3 Special Requests

Accept and act on special requests to interrupt display or processing, change the sequence of execution, or discontinue operation. Such facility would include pre-defined functions implemented by means of a special keyset (e.g., HELP key) or other means. It must be possible to disable these special request functions under programme control. *0*

2.01.4 Audio Input

Accept audio from microphone input to audio recorder for later playback to the user (student, teacher, author, researcher). *2*

2.02 Set Input Conditions

2.02.1 Define Input Limit

- Define a limit on the time and/or number of characters allowed for input, after which some other action is taken, e.g., user is interrupted and appropriate display is presented as defined by the programme. The limit may be based on results of calculation at execution time. *2*

- It is desirable that information regarding the input state at the time of interruption be returned to the programme, e.g., no characters, n characters, etc. The partial input must be available to the programme.

2.02.2 Time Input

Measure elapsed time of user input, e.g., from when he is permitted to enter information to when he sends a completion or "judge" character, or meets some other specified condition. A resolution of 1/4 second is desirable at level 2 with coarser resolution being acceptable at levels 0 and 1. *0*

2.03 User Alteration During Input

- Allow user to alter his current input message, e.g., cancel entire message, erase preceding characters, change, delete, and insert characters.

- At level 1 and higher, this facility also will apply to graphic components of the input message (points, lines, figures, etc.). In the case of alteration of graphic input, it would be desirable to provide a "replace command" which would cancel a preceeding delete in order to facilitate complex graphical construction. *0*

*Level*

2.04 Inter-User Communication

2.04.1 Accept Communication Message

Accept messages for transmission to one or more terminals to permit conversations between users (student, teacher, author, researcher). 2

2.04.2 Accept and Record User Communication Message

Establish a "common" area or file which can be written on and read from by all users. Anticipated uses include gaming and conversations between users, and studies of group interaction processes. Messages would normally consist of alphanumeric data, but if facilities exist for the manipulation of graphical data at the terminals, this mode should be permitted for inter-user communication. 2

2.05 Experimental Input Features

The following input control features are considered to be experimental and are recommended for inclusion in level X implementations:

2.05.1 Accept Audio for Analysis

- Accept audio from microphone input to digital (or analogue) processor for analysis by the programme. X

2.05.2 Accept Input from Special Devices

- Accept digitized input from specialized devices. It should be possible to identify such devices by means of logical names. X

## 3. *Process User Input*

### 3.00 General

The usefulness of a language for CAL applications will be determined to a major extent by its built-in facilities for processing user input. The language must provide for a wide range of input devices and make sufficiently flexible routines available to apply these devices in a variety of situations.

The CAL language should contain as embedded functions a sufficient range to handle most requirements for input processing. However, there must also be provision for user defined functions which can be invoked by the author in the same way as the built-in functions. Note that these user-defined functions must be constructed using commands and functions contained within the language. Otherwise the programmes are not transportable to other installations where the required level of the language has been implemented.

In the case of the embedded functions which are called automatically, a hierarchical ordering should exist such that if a condition is reached at a given point, further automatic input processing functions are not carried out. It is desirable that it be possible for an author to specify a hierarchy for his user defined functions.

Facilities for the manipulation of user input must be invocable without having to resort to complex programming techniques, (which, in fact, would make the facility unavailable to many of the users of a CAL system). Those input processing facilities which are used by a great majority of users might be called automatically unless they are "turned off" by the user.

In summary, input processing routines may be invoked in a programme through the following:

1. Automatic Input Processing Functions
   - An automatic input processing function is a procedure applied to all input unless otherwise specified by the author.

2. Defined Input Processing Functions
   - Several methods could be made available to the author to permit him to invoke a procedure:
     - A procedure (author or system defined) may be specified by the author to be called automatically as in 1 above.
     - In-line calls to a pre-defined procedure (author or system defined) for application to a single input or set of inputs.
     - In-line coding of a procedure.

Many of the functions described in this section involve string processing. These functions may be most efficiently implemented by specification of a set of coding conventions and operators such as those used in currently available string processing languages (e.g., COMIT, SNOBOL). In particular, it will be important for a user's input string (or part thereof) to be given a variable name to permit referencing for subsequent use in expressions and manipulations with other variables and strings.

3.01 Edit Input

The following editing functions should be available to an author by means of a switch setting. It should be possible for the author to specify a sequence of editing operations which will be carried out on all input. It is desirable that more than one such sequence be specifiable and

*Level*

selectable. In all cases, the unedited input string should remain available to the programme. This is of particular importance in the case of unrecognized responses.

3.01.1 Delete Characters

- Delete any specified character(s) from a designated character string. The resulting edited string will be "compressed" to remove the blanks so created unless otherwise indicated. *0*

- Specific instances of this function which may be invoked automatically should include:

1. Removal of multiple spaces
2. Removal of all format control characters (space, tab, carriage return, etc.)
3. Punctuation removal
4. Special character removal

3.01.2 Replacement

- Modify a character string by substituting specified replacement characters for designated characters on a one for one basis. *0*

- Modify a character string by substituting specified replacement substrings for all occurrences of designated substrings. *2*

- This replacement function shall apply also to substrings embedded in character strings.

Instances of this function which may be invoked automatically for input strings include:

1. Shift to upper or lower case *0*
2. Edit list substitution
   (a) Character for character *0*
   (b) Character string for character string *2*

*Level*

3.01.3 Extraction

- Extract only numeric, or only alphabetic characters from the designated character string. The function must permit the extraction of multiple fields from the same character string. *0*

- This function should be capable of being invoked automatically in the following form:

1. Extract numeric *0*
2. Extract alphabetic *0*

3.01.4 Normalize

Replace numeric strings with equivalent in normalized form.
For application to input editing, this function should be capable of being invoked in the following form: *0*

- Extract and normalize numeric

3.01.5 Encode

- Encode a character string into a phonetically reduced form so that phonetically similar spellings can be detected. *2*

- Provide binary flags to indicate the presence or absence of specified substring components or patterns in the student input text. These flags can then be used to perform Boolean logic in accord with the student answer. *2*

3.02 Match Input

Functions in this section are aimed at providing the necessary procedures to compare the student response with the data entered by the author. Flexibility is of utmost importance, but the author should have at his disposal automatic

routines which may be invoked in a matching hierarchy and applied to all student input until turned off. The author must be able to select (by means of a switch setting or other simple operation) which functions will be invoked and in what order. The following automatic matching functions should be available to the author:

- Exact match *0*
- Similarity match
  1. Based on a simple algorithm such as Damerau's* *0*
  2. Based on phonetic equivalence *2*
- Keystring or Keyword occurrence
  1. Single *0*
  2. Multiple strings *2*
     a) In specified order
     b) In any order
- Similarity keystring match (applied as above using an algorithm such as Damerau's) *2*
- Algebraic equivalence *2*
- Numerical equivalence within limits *0*
- System control word recognition (HELP, etc.) *0*

3.02.1 Character String Processing

.1 Attempt exact match, character for character of input string with string of characters prestored by author. *0*

.2 Attempt exact match and return percentage or number of characters matched. *2*

.3 Attempt exact match and return the location in the stored character string at which comparison first fails to an appropriate memory item. *2*

- - - - - - - - - - - - - - - -

*Damerau, F. A Technique for computer detection and correction of spelling errors. Comm. A.C.M., 7(3): 171-176; 1964.

*Level*

.4 Define string and single character line editing symbols to force match on arbitrary character in designated portions of string ("Wild" string or character symbols). *0*

.5 Attempt match of character string with multiple strings prestored by author, according to logic specified in the instruction. *0*

e.g. IF X = .31g OR 310 mg OR 0.31g

THEN.... ELSE....

This facility should include provision at level 2 for referring to a list of synonyms by means of a single label, or possibly by means of any word contained in the list of synonyms.

.6 Attempt match of character string with multiple key strings (or keywords). Match would occur only if all key strings (or keywords) existed in the input string. Provision would be included to specify order of occurrence as requirement for match. *0*

.7 Match for similarity on the basis of selected algorithms. (See 3.01.5). *0*

3.02.2 Numeric Processing

.1 Attempt exact match, recognizing equivalent integer numbers coded in different formats (e.g., 3,3.00, 3E+00, etc.). *0*

.2 Establish tolerances for anticipated numerical responses. A match would then occur if the number falls within the specified limits. *0*

.3 Permit author specified arithmetic operations on numeric expressions, either as entered by student, or assembled from student input and prestored (or generated) components. *0*

*Level*

.4 Permit author assigned numeric values for character strings, e.g., mg = ÷1000, g = X1, kg = X1000 — *2*

3.02.3 Algebraic Processing

.1 Evaluate an expression using predefined values. (Implementation at level 0 would be restricted to simple expressions and would not permit the use of parenthesis which should be permitted at level 2). — *0*

.2 Recognize equivalent expressions by assigning randomly selected values to variables (range may be specified) of student and author answers. Match occurs if answers are the same numerically within a specified tolerance. — *0*

.3 Identify and separate variables in an algebraic expression. — *2*

3.02.4 Graphics Processing

.1 Compare input coordinates from light pen, touch sensitive tablet, or other graphic input device with regions defined by author (either locally or globally). — *1*

.2 Match pattern of points from a graphic input device with a prestored pattern. — *1*

.3 Match pattern of points from a graphic input device with a prestored pattern giving consideration to the order in which the points were entered by the user. — *X*

.4 Match pattern of points according to a specified algorithm (e.g., match sine wave, or other pattern defined by a mathematical function). — *X*

3.03 Characterize Input

Functions in this section are aimed at providing the author with as much information as possible regarding the input. The functions also should facilitate the recording of data associated with input characteristics. As with the functions in 3.01 and 3.02, it should be possible for the author to involve certain automatic functions. Some of these automatic functions (for example setting response matching indicators) would be invoked by action taken regarding automatic functions in 3.01 and 3.02. Automatic functions available should include:

- Set response match indicator (according to parameters established in 3.02).
- Indicate percentage of characters matched in input string.
- Response terminated by "time-out".
- Number of characters of input received.
- Location of last character matched in input before failure.
- Number of times student has matched on same item.
- Automatically incremented counters for "right", "wrong", neutral, unanticipated, timed out responses.

3.03.1 Delimit Fields

.1 Provide for referring to data of the following types as literals, labelled constants, and variables.

- Variable-length strings *0*
  - Explicit naming *0*
  - Implicit naming *2*

Level

- Numerical Data
  - -Integer 0
  - -Floating Point 0
  - -"Decimal strings" (e.g., 172.63 without conversion to floating point) 0
- Bit Data 2
- Matrix Data X

.2 Delimit and name substrings on the basis of character type, or absolute or relative location in the string. 2

.3 Determine the location(s) of a given character or substring in the student input string. 2

.4 Count the number of times a given substring appears in a string and store the count as a named variable. 2

3.03.2 Classify Input

.1 Identify an input string as being a question. X

.2 Identify unanticipated responses as such. 0

3.04 Respond to Unrecognized Input

These functions provide predefined implicit branching which may be initiated when a response has not been recognized.

3.04.1 Error Messages

Provide a list of standard (may be author specified) error messages, and/or a default message to student when author makes no special provision for unrecognized response. 0

3.04.2 Multiple Occurrences

Allow author to specify multiple unrecognized response actions, so that the next item from the

*Level*

list is executed in sequence, each time an unrecognized response is encountered. The maximum number of any of these actions should be specifiable. *2*

- Implementation at level 0 should be limited to multiple branch lists.

3.04.3 Formatting

Provide for specification of special formatting of unrecognized inputs for later inspection and analysis by the teacher or author (see 6.01). *2*

3.05 Respond to Student Requests

3.05.1 - Process students request for modification of sequences of pedagogical units. Check for prestored special words (e.g., HELP, COMMENT, CALC, BRANCH, GLOSSARY, etc.) and initiate appropriate action. (see 4.04) It should be possible for an author to disable selectively the functions specified by these special words. *0*

- At level 0, the implementation of this facility may be limited.

3.05.2 - Process student's questions concerning course content. *X*

4. *Identify Locations in Programme*

4.00 General

The language must include labelling facilities to permit the author to define the path to be taken through the programme, or sections thereof, either on a pre-determined basis, or according to interpretation of results obtained during execution of

the programme.

These labelling facilities should be sufficiently flexible to permit easy identification of "micro elements", larger structures such as "frames", and also self contained units such as "concept blocks".

The language should permit the incorporation of other programmes or parts of other programmes by an author in his main programme and automatically resolve any conflict between labels internal to these sections of coding and the main programme.

4.01 Label a Statement

- Label an identifying point to which a branch may be made. A label may be assigned to any statement in the programme, but it should not be a requirement that every statement have a label. *0*

- Alphanumeric labels shall be permitted and it should be possible to treat them as integer variables. *0*

- It is desirable to permit the construction of labels by the programme. *X*

4.02 Identify by Condition

Identify a statement by a condition or aspect true to more than one statement, e.g., the statement type (operation performed), the position within a hierarchy, a tag or other descriptor, the initial characters of text, or any transformation applied to the text. *2*

4.03 Establish a Restart Point

Establish a restart point at which execution will begin again if a disconnection occurs before a *0*

sequence is completed, whether by accident or intention. The restart conventions must be completely defined. The exact status of internal registers and all external devices must be dealt with appropriately. The author should have the option of deciding what operations are to take place on restart, but default conventions should exist.

4.04 Identify by Reuseable Label

Identify a point by means of a label taken from a set of author defined reuseable labels. The point referred to by the labels in this set will be determined by the most recent definition of the particular label in the programme. (i.e., each time a label is defined by an author, there will be an automatic re-assignment of value. This new value will apply to all references to the given label in all programme sequences which follow until a new definition for the label is encountered).

A basic standard set of such labels shall be made available (e.g., HELP, GLOSSARY, etc.) which may be added to or changed by the author. *0*

Student access to such labels should be provided, but such access should be switch controlled (may be turned off by the author at any point in the programme).

5. *Sequence Execution*

5.00 General

The language must provide very extensive and flexible branching facilities. It should be

possible to exit from the main sequence stream to another part of the programme, or in fact to a different programme, and return with counters and status switches unchanged or modified as specified by the author. Where applicable, the status of external devices must be remembered and dealt with appropriately.

5.01 Branch to Label

Branch to a location with a label defined under the terms of Section 4. (The label may be either internal or external and it may be a variable whose value is assigned by the programme at execution time). *0*

5.02 Branch to Condition Referenced

- Branch to a location defined in relation to: *0*

  - present statement location (e.g., here plus two).
  - the current state of execution (e.g., last executed display, or last executed request for response).
  - condition or aspect or other statements (e.g., next statement in the listing identified as a "problem").

5.03 Branch to Subroutine

- Branch to subroutine, i.e., branch to label and save information defining location to which execution should return after the subroutine is completed. (Usually location of next statement following branch).

*Level*

- Many levels of nesting should be permitted. (A practical limit of 10-20 has been suggested as sufficient). *0*

- Fully re-entrant subroutines should be possible.

It should be possible to save the content of specified counters and status switches and have them automatically restored on return. (The save specification should provide for saving all, without having to specify in detail).

5.04 Return from Subroutine

- Return from subroutine to the location saved before the subroutine was entered. If switches were saved by the calling instruction these should be automatically restored in the return operation. The subroutine may return status information which it must be possible to add to the switch settings as restored. *0*

- In the case of nested calls, it should be possible to return directly to the main line calling programme and to clear the address stack automatically. *0*

- It should be possible to return to any given step in the calling sequence in a multi-level call with appropriate corrections to the calling address stack, being made automatically. *2*

5.05 Repeat Conditional on Logical Expression

Repeat designated section until some condition is met:

- Iterate to a numerical or single variable limit. *0*

- Iterate to a limit which may be any arithmetic expression. *2*

*Level*

- Iterate until non-numeric condition is satisfied (e.g., switch set, time limit exceeded, specific input received). *2*
- Both forward and reverse indexing should be allowed. *2*

5.06 Branch Conditional on Logical Expression

- Branch conditional on the value of a logical expression. This expression may refer to the values of numeric or string variables in which current user input or records of interaction may have been saved and manipulated.

- This branching operation may be dependent on:

  - programme variable, counter, etc. *0*
  - user input *0*
  - record of individual user interaction *0*
  - cumulative interaction record of programme use, either alone or in conjunction with record of individual user interaction *2*

- Variables should be allowed for labels in branching statements. *0*

5.07 Branch on Computed Value

Branch to a line number, numeric label, subscripted label name, or to the nth element in a set of labels, statements or programme parts, using as a pointer the value of a variable or expression. *0*

5.08 Select from List

- Select next statement from list according to a rule such as: *2*

  - Go to a label selected from a list either in sequence or randomly.

- Assign a value to a numeric or character string variable selected from a list either in sequence or randomly, e.g., X = (CALCIUM, FERRIC, FERROUS, CADMIUM) RANDOM. Here the variable X will be assigned the character string values indicated in random order, without repetition, on successive executions of this statement.

- Specify attributes that terminate the list selection operation after a given condition is met. Store in memory the position of each item in the list, the number of times each item is selected, and the total number of selections made.

e.g., GO TO (A,B,B,C) RANDOM 5 TIMES. Here the three different labels will be selected randomly, with B more heavily weighted, until after 5 selections have been made.

e.g., X = (1, 2, 3, 4, 5) UNTIL RIGHTS = 0.8 * TRIES OR 10 TIMES. Here the variable X is assigned the successive values indicated until the variable RIGHTS is 80 percent of the number of tries. If this condition is not satisfied, execution ceases after the list has been repeated 10 times.

- Retain in list and repeat the selections in which student answers are flagged as "wrong".

e.g., X = (1, 2, 3, 4) RANDOM 6 TIMES REPEATING WRONGS TO 2 TIMES.

- Specify, or place a limit on the number of times any one item can be selected.

e.g., X = (CALCIUM, STRONTIUM, BARIUM) RANDOM (2,2,3) TIMES.

*Level*

e.g., GO TO (12.1, 12.2, 12.3, 12.4) RANDOM ELIMINATING (1, 2, 3, 4) WHEN (A=2, B=3, C=3, D=2) TO 10 TIMES. Here the first item (label 12.1) is removed from the selection list when the externally-defined variable A reaches the value 2, and so on.

- Assign to each item a probability for weighting in random selection.

e.g., X = (MANNY, MOE, JACK) RANDOM 10 TIMES WEIGHTS (2, 1, 2)

- Note that wherever a random number generator is used to control an operation, as in this function, it should be possible to switch to a controlled sequence to facilitate debugging.

5.09 Branch on Restart of Execution

Branch to the designated position of a programme segment in which the user last signed off or otherwise stopped using the programme. The branch may be conditional on information about the user or the condition of the programme, including the way in which the user signed on (requested) the programme on this occasion. (See Section 4.03). *0*

5.10 Predefined Implicit Branching

Implicit branching should be built into the language which will be invoked under specific conditions such as occurrence of a match, or a failure to match a response, etc. Author override must be possible. *0*

6. *Record and Manipulate Data*

6.00 General

The language must provide facilities to

record and manipulate a wide range of data generated by and necessary to the operation of a CAL system. Records of individual student's performance can be used and in fact are required in the control of presentation or the prescription of course materials to the student if individualized instruction is to be achieved. Records of student progress (both group and individual) must be made available to the teacher, preferably in a standard format which can be readily understood.

Records must be kept of programme performance to provide necessary data for use in validation and updating of programmes.

Basic elements must be available within the language for the manipulation of data. These elements will permit in line coding of procedures as well as definition of new macro procedures which the author may then invoke by the same methods that apply to automatic functions.

In simulation and gaming applications, it is particularly important that the author be able to manipulate values and variables within the language. In such cases, it is necessary to use data input from terminals as well as those prestored or generated with sophisticated computational facilities.

6.01 Record User Data

- Save data on user interaction (e.g., student performance) so that it is available to the programme (e.g., instructional logic of the author).

- Of particular importance in CAI applications is the recording of unanticipated responses.

Level

6.02 Record Programme Data

Save data on programme interaction with various users and make it available to the programme logic. This permits self-modification of the programme to be prescribed on the basis of the data common to any or all users to date. It is anticipated that the implementation of this facility at levels 0 and 1 may be more limited than at level 2. 0

6.03 Record Community Data

Save information for on-line reference and change by any user of the programme. (See 2.04.2) X

6.04 Record Data for Later Summary

Store data, probably on a peripheral device, for later analysis, normally in batch mode. 0

6.05 Define Data Analyses and Report Formats

- Standard specifications for data analysis and summary for later inspection which are available to the author by means of parameter switches. 0

- A selection of such standard recording formats should be available which will permit the identification of the data recorded with the particular situation. 0

- The author should be permitted to describe his own reporting formats and control these by means of parameter switches in the same way as the standard formats are invoked. 0

*Level*

6.06 Manipulate Values

6.06.1 - Execute computations involving data of the following types, in the form of literals, labelled constants, and variables:

- Real *0*
- Integer *0*
- String *0*
- Bit *2*
- List *X*

6.06.2 Execute computations involving matrices *2*

6.06.3 Computations and manipulations should include the following operators:

1. Arithmetic

- Substitute *0*
- Add *0*
- Subtract *0*
- Multiply *0*
- Divide *0*
- Square Root *0*
- Raise to a power (integer only) *0*
- Standard Mathematical Functions (Library Routines) *2*
- User defined functions *2*

2. String

- Concatenation and indexed extraction *0*
- Grouping (as in SNOBOL) *0*
- Length (Function the value of which is the string length) *0*
- Back-referencing and anchored scan (As in SNOBOL) *2*

*Level*

3. Bit

- Set/Reset *2*
- Complement *2*
- Rotate left and right *2*
- Compare two bit strings *2*
- Boolean Functions *2*

4. List

- Elementary list processing operations *X*

6.06.4 Generate Random Numbers

1. Uniform Distribution *0*
2. Normal Distribution *2*
3. User Defined Distribution *X*

In all cases, it shall be possible to replace the random number sequence by a known sequence to facilitate debugging.

6.07 Control of Record Operations

All recording operations shall be subject to a switch which may be set during execution to facilitate testing and debugging a programme. *0*

7. *User Tools Available within the Language*

7.00 General

The language should contain a simple computational facility which is readily available to author, teacher, or student. In addition, the language should include within its structure the possibility of providing the user access to other languages which may be available within the computer system on which it is implemented. It should not be necessary, for example, to exit

from a CAL programme teaching FORTRAN in order to run a test FORTRAN programme. It should be possible to access the FORTRAN compiler directly, even to the extent of returning results to the CAL programme.

It is also very desirable that the CAL language should provide for access to interactive languages such as APL, BASIC, CPS, FOCAL, etc.; which might be available within the system.

It is to be noted however, that no CAL programme should call on routines written in assembly language, or in another high level language unless it cannot be avoided. Any programme which does must contain a warning to users and must include a detailed description of the function written in the external language.

7.01 Desk Calculator

Desk calculator or other simple computational aid such as a mathematical expression evaluator with assign statement and variables which may be defined for the purpose in the main programme. *0*

8. *Coding Convenience*

8.00 General

All possible coding conveniences should be made available for the preparation and presentation of materials using a wide range of instructional strategies. The coding conveniences must take into account that many of the users of the language will not be sophisticated computer users. However, the coding conveniences should not be such as to "interfere" with access to the system by the more computer-oriented user.

*Level*

8.01 Modification Preliminary to Execution

Specifications for automatic modification of the content or arrangements of character strings during processing prior to execution (compilation, scanning, testing) to be applied to (selected) users who follow. *2*

8.02 Assign Attributes

Assign attributes to objects by which they may later be identified and/or characterized for answer processing or other decisions. Such a facility would include the ability to equate, to assign hierarchy, and to name by group. *2*

8.03 Precoded Procedures

- Precoded procedures, i.e., standard routines which can be called by the user to operate on lists of materials, e.g., a subroutine which can be applied to lists named as arguments. *0*

- It must be possible to separate logic and data so that procedures can be selected and/or prepared according to a given teaching strategy and can then be applied to data supplied for a specific application. *0*

- Routines should be available to assist in the generation of graphics. These might include: *1*

  - Line or vector generators
  - Circle generators
  - Special alphabet generation routines (See 1.02.3)

8.04 Comments or Remarks

- Comments or remarks inserted for the use of the author, editor or programmer but not executed must be permitted. Such comments or *0*

remarks should be readily available to the author during debugging.

- A facility for recording student comments should be available during execution. *0*

9. *Guide to Preparation of Detailed Specification and Implementation*

9.01 Preparation of Programme

The preparation of the computer copy of the programme shall be possible using a wide range of input/output devices. Both on-line and off-line devices should be permitted. Use of all storage media (e.g., cards, tape, etc.) shall conform to applicable standards.

9.02 Format of Programme

The format of the programme should be relatively "free" but should be such as to facilitate "built in" documentation. The format should take into account that many of the users will not be "computer-oriented".

9.03 Diagnostics and Test Features

The language should provide for automatic assistance in testing, e.g., diagnostics, programme traces, test with interrupts, etc.

9.04 Storage of Programme and Data

All storage of instructional material and logic ("off-line") should conform to applicable standards and to a limited range of formats to facilitate exchange of materials between different installations.

9.05 Author Editing Mode

Devices available to the author for editing and updating should be the same as those used for programme preparation.

9.06 Author Editing Features

Provision should be made for on-line (or off-line in some cases) editing of programme and data files. In the case of on-line operations, there should be easy transition from editing mode to debugging mode.

9.07 Author Test Features

- Provisions for testing programme during and following execution, interrupting on specified conditions, examining values of variables or log of interaction, in detailed or summary form, modifying values or the programme itself, and resuming execution at desired location.

- Checking routines and test programmes should be available to assist in the identification of errors.

9.08 User Registration

- Provision must be made for the registration of students, authors, and other users to allow access to the system to allocate storage space for user records, and to account for resources used.

- The registration facility should include a means of restricting access to the system facilities, e.g., students who can take a course, author who may inspect and revise it, teacher who may inspect student data, etc.

9.09 On Line-Assistance to User

Facilities should be incorporated in the system to provide on-line assistance to the user such as:

- Interrupt the user to provide immediate assistance on line.

- Permit operator intervention to transfer a student to some location in the programme by means of an immediate GO TO LABEL instruction.

- The operator should have the capability to monitor an individual student's interaction with the system.

- To facilitate effective operator assistance, it should be possible to generate a cross reference listing and variable map for any programme.

9.10 Record Handling

The system should provide record handling routines to facilitate the manipulation of data gathered during operation. For example, where standard recording formats are used routines should be available for producing summaries of programme performance, or summary reports of an individual student's performance.

9.11 Documentation

- The detailed specification should include recommended formats for documentation of course materials. It is particularly important that a potential user of a computer based course have access to a sufficiently detailed description of the materials to determine whether or not they meet his needs. Because of the different classes of users, more than one level of documentation may be required.

- If there is to be an effective exchange of computer based instructional materials, there is a requirement for a mechanism for indexing of CAL materials so that their existence can be made known to organizations other than the group which has produced the particular materials. Standard documentation formats are vital to such an indexing system.

- A further aspect of documentation relates to the programming reference manual and author's manual. All too often, these manuals are written in a language which is incomprehensible to all but the computer expert. The manuals must be easy to use and must take into account that the majority of users will not be "computer-oriented".

## Appendix A

Documents Presented to the Working Panel

1. VAULT Concepts and Facilities. W. Birtch, R. R. Jordan, E. W. Romaniuk, --The University of Alberta.
2. CAN-4, An Advanced Author Language for CAI, Computer-Based Testing and Psychological Experimentation PDP-9 Implementation. Stacy Churchill, Lars [illegible]ss, and William P. Olivier, --The Ontario Institute for Studies in Education.
3. Course Author Languages for Computer Aided Instruction and Curriculum Materials Preparation: OISE Developments 1967 - 1970. Stacy Churchill, --The Ontario Institute for Studies in Education.
4. Necessary Characteristics of a Language for Successful Computer Assisted Instruction. Brian V. Funt, --Simon Fraser University.
5. Summary of Coursewriter II and APL following Zinn's list. Steve Hunka, --The University of Alberta.
6. Note sur un langage orienté vers l'enseignement avec ordinateur. François Labrousse, --Service de l'Informatique Ministère de l'Education du Québec.
7. CAI Language: Some Preliminary Considerations. Stephen K. Lower, --Simon Fraser University.
8. A functional Evaluation of Coursewriter III. Stephen K. Lower, Gillian Arsenault, --Simon Fraser University.
9. Learning about Computers Via an Interactive Programming Language. R. S. McLean, L. D. McLean, --The Ontario Institute for Studies in Education (Presented at IFIP World Conference on Computer Education 24-28 August, 1970, Amsterdam, The Netherlands).
10. Recommendations for a Computer Assisted Learning Language. Rosemary L. Newkirk, --The University of Western Ontario.

11. Aspects of CAL Languages (Zinn, 1969) as related to CAN-6. W. P. Olivier, --The Ontario Institute for Studies in Education.

12. Recommendations for CAL Language Specifications. G. H. Roid, --McGill University.

13. User's Guide to "MULE", McGill University Language for Education, A Computer-Assisted Instruction Language. Gale H. Roid, --McGill University.

14. SCROLL An Interpretive Language for CAI Programmes. --The University of Calgary.